8. Типове данни за цели и реални числа

Проф. д-р Емил Хаджиколев

1. Типове за цели числа
2. Типове за реални числа
3. Съвместимост на типовете
4. Преобразуване на типове
5. Аритметични оператори
6. Оператори за сравнение
7. Стандартни математически функции

Брой стойности в целочислените типове

- Броят на различните числа, които могат да бъдат представени с целочислените типове зависи от броя на битовете им и се определя като 2 на степен броя на битовете.

размер	**брой**	**степен**
8 бита	256	$= 2^8$
16 бита	65 536	$= 2^{16}$
32 бита	4 294 967 296	$= 2^{32}$ (около 4 милиарда)
64 бита	18 446 744 073 709 551 616	$= 2^{64}$ (много)

Целочислени типове със знак и без знак (1)

- **Типове със знак** се означават с ключовата дума **signed**, която може да се пропусне (тя е по подразбиране):
  - signed char;
  - signed short;
  - signed int;
  - signed long;
  - signed long long;
- **Типове без знак** се означават с ключовата дума **unsigned**:
  - unsigned char;
  - unsigned short;
  - unsigned int;
  - unsigned long;
  - unsigned long long;

Целочислени типове със знак и без знак (2)

- При типовете със знак, **знакът се определя от първия бит на числото.** Ако стойността му е:
  - 1 – минус;
  - 0 – плюс;
- В следващата таблица са представени дефиниционните области на основните типове:

тип	**стойности в интервал**	
short	[-32 768, 32 767]	= $[-2^{15}, 2^{15}-1]$
unsigned short	[0, 65 535]	= $[0, 2^{16}-1]$
int, long	[-2 147 483 648, 2 147 483 647]	= $[-2^{31}, 2^{31}-1]$
unsigned int, unsigned long	[0, 4 294 967 295]	= $[0, 2^{32}-1]$
long long		= $[-2^{63}, 2^{63}-1]$
unsigned long long		= $[0, 2^{64}-1]$

Литерали за цели числа в C++

- Литералите са неименувани константи (на които не задаваме тип, но), които автоматично получават тип при компилиране.
- По подразбиране целочислените литерали получават тип int. Литералите за char и short се дефинират чрез типа int; за да се укаже, че литерала е от тип long, след числото се записва буквата ‘L’ (или ‘l’ – този вариант не е много четим); за long long – LL. Съответно, различните литерали заемат различен размер в паметта…

```
char age = 20;
short year = 2015;
int counter = 33000;
long identificator1 = 5L;
long long identificator2 = 5LL;
```

Представяне на числата в различни бройни системи

- В C++ може да се задават числа в различни бройни системи – с основа 2, 8, 10, 16.
- Стандартният начин за показване на стойностите е в десетична бройна система, без значение от начина на задаването им.
- Особености при представянията на числата в различни бройни системи са:
  - двоична – започват с 0b (или 0B); използваните цифри са {0, 1};
  - осмична – започват с 0; използваните цифри са {0, 1, 2, 3, 4, 5, 6, 7};
  - десетична – използваните цифри са {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  - шестнайсетична – започват с 0x (или 0X); използваните цифри са {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}; цифрите над 9 може да се задават с малки или главни букви.
- Тези правила са илюстрирани в следващия примерен код:

Литерали за цели числа в различни бройни системи

```
int binNumber = -0b110;   // основа 2
int octalNumber = 037;    // основа 8
int decimalNumber = 19;   // основа 10
long hexNumber = -0xFE;   // основа 16

cout << "binNumber->" << binNumber << '\n';
cout << "octalNumber->" << octalNumber << '\n';
cout << "decimalNumber->" << decimalNumber << '\n';
cout << "hexNumber->" << hexNumber << '\n';
```

Резултат:

binNumber -> -6

octalNumber -> 31

decimalNumber -> 19

hexNumber -> -254

Типове за реални числа

Типовете за **реални числа са два основни вида**:

- **реални числа с плаваща запетая** – в езиците за програмиране обикновено имат две реализации – примитивни типове float (32 бита) и double (64 бита) ;
- **реални числа с фиксирана запетая** – не всички ЕП поддържат такъв примитивен тип:
  - Някои ЕП (като C#, SQL и др.) поддържат вграден примитивен тип decimal;
  - Други ЕП (като C++, Java и др.) езици имат допълнителни библиотеки (библиотека decimal в C++, клас java.math.BigDecimal в Java), които могат да се ползват за работа с реални числа с фиксирана запетая.

Реални числа с плаваща запетая

- Реализират стандарта IEEE 754;
- Има ограничения в броя на използваната памет – 32 или 64 бита;
- Първият бит определя знака на числото – 0 за ‘+’; 1 за ‘–‘.
- В следващите битове се записват:
  - порядък p (експонента) – цяло число (включително и знака му);
  - мантиса M – реално число между 1 и 2.
- Абсолютната стойност на реалното число е $M*2^p$. В зависимост от порядъка, битовете за мантисата са плаващ брой, откъдето идва и името на числата с плаваща запетая (floating point).
- Очевидно (?!), не всяко математическо реално число от реалния свят може да бъде представено по този начин. Поради това често се получават различни грешки при закръгляванията на такива числа.

Реални числа с фиксирана запетая

- При тях десетичната запетая не променя позицията си.
- Реалното число се представя като две цели числа (понякога без ограничения в битовете) – за цялата част и за частта след десетичната запетая;
- В някои езици има ограничения за общия брой символи на числата, в други могат да се записват много големи числа.
- Не се получават грешки при закръгляване (освен ако не се прехвърлят границите – при езиците, в които има граници).

Типове за реални числа с плаваща запетая в C++

- Типовете за реални числа с плаваща запетая в C++ са float (32 бита), double (64 бита) и long double (което в повечето реализации не се различава от double и също има 64 бита; може, обаче, да има повече битове – обикновено 80);

тип	**представяни числа**	**максимален приблизителен брой цифри**
float	$\pm 3.4 * 10^{\pm 38}$	7
double	$\pm 1.7 * 10^{\pm 308}$	15

Литерали за реални числа в C++

- Реалните числа се записват с цифри за цялата и дробната част, а десетичната запетая се означава с ‘.’ (точка).
- За да се укаже, че литерал е число от тип float, се записва f (или F) в края. Ако след числото не е указан символ, се приема че то е от тип double.
- При задаване на литералите може да се укаже порядък чрез символ E (или e – латински) последван от степента. Например, 1.2E2 е числото $1.2*10^{2}$, -2.5E-3 – $-2.5*10^{-3}$.

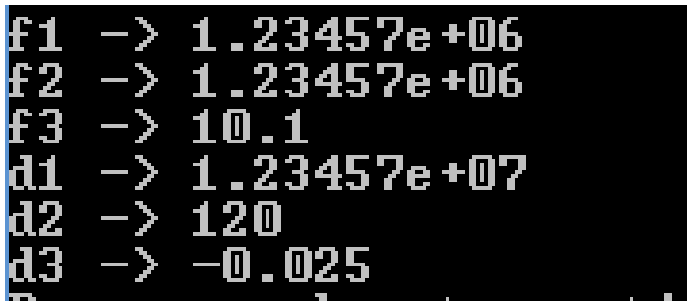
Пример – литерали за реални числа

```
float f1 = 1234567.1f;
float f2 = 1234567;
float f3 = 10.1f;

cout << "f1 -> " << f1 << '\n';
cout << "f2 -> " << f2 << '\n';
cout << "f3 -> " << f3 << '\n';

double d1 = 12345678.12345678;
double d2 = 1.2E2;
double d3 = -2.5E-2;

cout << "d1 -> " << d1 << '\n';
cout << "d2 -> " << d2 << '\n';
cout << "d3 -> " << d3 << '\n';
```



Специални стойности за реалните числа с плаваща запетая

- Според стандарта IEEE 754, поддържан от C++, към реалните числа с плаваща запетая са добавени 3 специални стойности, равностойни на другите стандартни числа:
  - inf (Infinity) – безкрайност;
  - -inf (-Infinity) – минус безкрайност;
  - nan (NotANumber) – не число.
- Има няколко случая, в които се получават такива стойности, някои от които показваме в следващия код:

```
1 / 0 -> inf
-1 / 0 -> -inf
0 / 0 -> -nan(ind)
```

```
double one = 1.0;
double zero = 0.0;
cout << one << " / " << zero << " -> " << one / zero << '\n';
cout << -one << " / " << zero << " -> " << -one / zero << '\n';
cout << zero << " / " << zero << " -> " << zero / zero << '\n';
```

Съвместимост на типове

- Два типа са съвместими, ако обхватът (възможните стойности) на единия е подмножество или съвпада с обхвата на другия.
- При числовите типове може да се каже че:

  char < short < int < long < long long < float < double

- При сложните типове, при които имаме наследяване, типът наследник притежава всички характеристики на типът е родител и съответно се явява негово разширение или над-тип.

Преобразуване по тип

- Преобразуването по тип е понятие, свързано със съвместимостта между типове.
- Може да се преобразува величина от един тип в друг, ако двата типа са съвместими.
- Преобразуването от под-тип към над-тип (напр., short в int) не води до загуба на точност и се извършва неявно, без да е изрично указано.
- Ако се преобразува величина от над-тип към под-тип, може да се получи загуба на точност, ако стойността е извън обхвата от множеството от възможни стойности на под-типа. Преобразуването в такъв случай може да се извърши само със специален **оператор за преобразуване по тип**

  (<тип>) <величина>,

  чрез който се указва преобразуване на стойност от някакъв тип към зададения в скобите тип.
- В следващия пример са показани възможни, но не винаги коректни преобразувания на променливи от примитивни типове.

Преобразуване по тип - пример

```
short s = 43;
int i = s;   // неявно преобразуване по тип
float f = i; // неявно преобразуване по тип

// i = f;    // не може неявно да се преобразува
i = (int)f;  // но може да се преобразува явно от float към int


// Следват примери за преобразуване със загуба на точност
i = (int) 35400.9; // Преобразуване на double до int, със закръгляване надолу
s = (short)i;      // Преобразуване на int до short

cout << " i -> " << i << '\n';
cout << " s -> " << s << '\n';
```

```
i -> 35400
s -> -30136
```

Аритметични оператори

Операндите на аритметичните оператори са числа, резултатът също е число.

Оператор	Име	Примери
+	събиране	a+b
-	изваждане	a-b
+	унарен плюс	+a
-	унарен минус	-a
*	умножение	a*b
/	деление	5/1→5, 5/2→2, 5/3→1, 5/4→1, 5/5→1
%	остатък при целочислено деление	5%1→0, 5%2→1, 5%3→2, 5%4→1, 5%5→0
i++ ++i	унарно увеличаване с единица (increment)	int i= 2; i++; i→3; ++i; i→4;
i-- --i	унарно намаляване с единица (decrement)	int i= 2; i--; i→1; --i; i→0;

Оператори ++ и --

- Унарните оператори ++ и -- може да се задават както преди, така и след променлива, чиято стойност трябва да се промени с единица:
  - Когато операторът е преди променливата – **++i, --i – първо се прилага операторът** и след това променливата участва в друг израз (ако има). (Аналогично на унарните минус и плюс – -i, +i.)
  - Иначе – **i++, i-- – променливата първо участва в друг израз (ако има)** и след това стойността ѝ се увеличава/намалява с единица.
  - (Ако не участват в сложен израз, използвания вариант няма значение.)

Оператори ++ и -- - пример

```
int i = 1;

cout << " i++ -> " << i++ << '\n';  // 1
cout << " i -> " << i << '\n';      // 2
cout << " ++i -> " << ++i << '\n';  // 3
cout << " i -> " << i << '\n';      // 3
```

Особености при аритметичните оператори

- Ако единият операнд е от целочислен, а другия от реален тип, резултата е реално число.
- При деление на цели числа, резултатът е цяло число.
- Ако е необходимо при деление на две цели числа да се получи реално, трябва да се преобразува поне единият операнд до тип за реално число.

```
int i = 10;
int j = 3;

cout << i << " / " << j << " = " << i/j << '\n';
cout << i << " / (double)" << j << " = " << i / (double)j << '\n';
```

```
10 / 3 = 3
10 / (double)3 = 3.33333
```

Оператори за сравнение

Операндите на операторите за сравнение са числа, а резултатът е булева стойност.

Оператор	Име	Примери с литерали	
==	равно	1==2→false	2==2→true
!=	различно	1!=2→true	2!=2→false
<	по-малко	1<2→true	2<2→false
>	по-голямо	1>2→false	2>2→false
<=	по-малко или равно	1<=2→true	2<=2→true
>=	по-голямо или равно	1>=2→false	2>=2→true

Математически константи и функции в C++

- Достъпни са в библиотеката cmath;
- За да ползваме константите (M_PI и неперовото число M_E), преди включването на библиотеката указваме директивата
  #define _USE_MATH_DEFINES;
- Има няколко групи функции:
  - тригонометрични;
  - хиперболични;
  - експоненциални и логаритмични;
  - степенуване и коренуване;
  - закръгляване;
  - и др.

Функции и константи – Пример (1)

```
#define _USE_MATH_DEFINES  // включване на математически константи

#include <iostream>
#include <cmath>           // математически функции

using namespace std;

// Отпечатване на константите от класа Math
void constants() {
    cout << M_PI << '\n'; // пи - 3.14159
    cout << M_E << '\n';  // неперово число - 2.71828
}
```

Функции и константи – Пример (2)

```
// Основни числови функции
void mainFuncs() {
    cout << abs(-6) << '\n';    // 6 - абсолютна стойност
    cout << log(M_E) << '\n';   // 1.0 - логаритъм от число зададено като параметър при
    основа E
    cout << exp(1) << '\n';     // 2.71828 - степен 1-ва (в примера) на неперовото число
    cout << sqrt(4) << '\n';    // 2.0 - корен квадратен от параметъра
    cout << pow(4, 2) << '\n';  // 16.0 - повдига първия параметър на степен втория
    cout << pow(27, (double)1 / 3) << '\n'; // 3.0 - и коренува (корен трети от 27)
    cout << fmin(3, 5) << '\n';  // 3 - минималното от две числа
    cout << fmax(3, 5) << '\n';  // 5 - максималното от две числа
}
```

Функции и константи – Пример (3)

```
// Закръгляване на числа
void roundFuncs() {
    cout << ceil(4.3) << '\n';  // 5.0 - закръгляване към по-голямото цяло число
    cout << floor(4.9) << '\n'; // 4.0 - закръгляване към по-малкото цяло число
    cout << round(4.6) << '\n'; // 5.0 - закръгляване към по-близкото цяло число
    cout << round(4.4) << '\n'; // 4.0
    cout << round(4.5) << '\n'; // 5.0
}
```

Функции и константи – Пример (4)

```
// Тригонометрични функции - работят с радиани - 360 градуса = 2*PI радиана
void trigonometricFuncs() {
    cout << sin(M_PI / 2) << '\n';  // 1.0 - sin от ъгъл, зададен в радиани
    cout << cos(M_PI / 2) << '\n';  // трябва да е 0 - cos от ъгъл, зададен в радиани
                                    // реално се показва много малко реално число
                                    // поради стандартните грешки при работа с реални числа

    cout << tan(M_PI / 2) << '\n'; // безкрайност-тангенс от ъгъл, зададен в радиани
    cout << sin(M_PI / 2) / cos(M_PI / 2) << '\n'; // == tan(PI/2)

    cout << cos(M_PI / 2) / sin(M_PI / 2) << '\n'; // 0 - за котангенс няма функция
}
```

Функции и константи – Пример (5)

```
// Извикване на дефинираните функции
int main()
{
    constants();           // константи
    mainFuncs();           // основни функции
    roundFuncs();          // функции за закръгляване
    trigonometricFuncs(); // тригонометрични функции
    // radiansAndDegrees(); преобразуване на радиани в градуси и обратно

    return 0;
}
```